HYPER

SCALING

ROCK RMS

Introduction

As digital ministry continues to grow, scaling your technical infrastructure becomes more and more important. Larger organizations may have unique needs to handle peak loads. The goal of this guide is to provide you with instructions and tips for scaling Rock to meet your needs.

Rock Web Farm

Large organizations soon reach the point where a single server will no longer be able to support peak loads. When this happens, the need for a server cluster becomes evident. Clusters, however, add several unique challenges, and their implementation must be approached with a healthy amount of planning and strategy.

The goal of this chapter is to provide an organization with best practices on how to implement a server cluster in a Rock environment and to cover the web farm features built into Rock.

Architecting Your Server Cluster

Scaling Directions

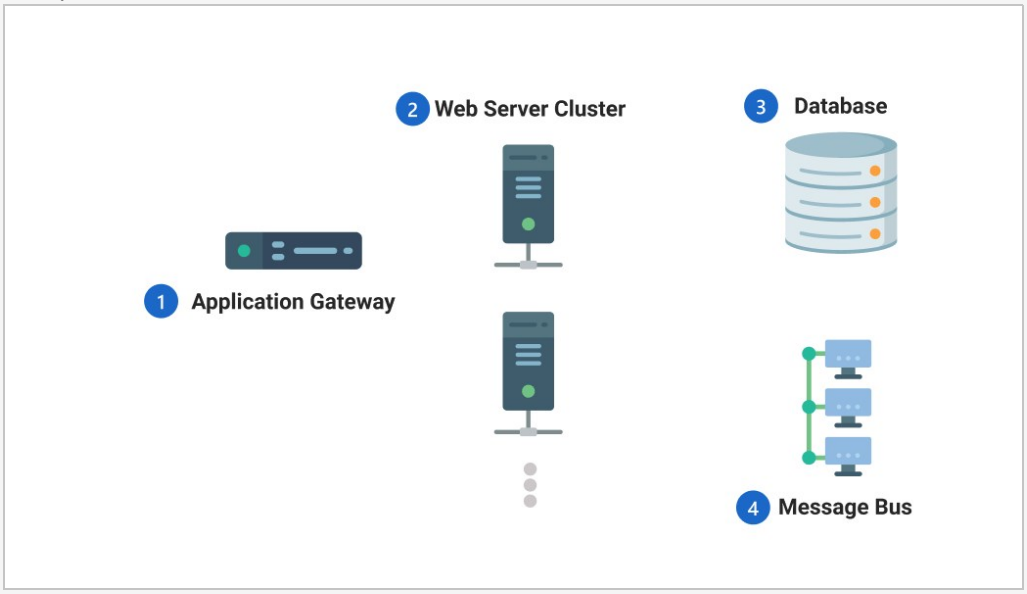
You have two directions when it comes to scaling your environment. Scaling up, sometimes referred to as scaling vertically, means that you increase the size of your server by adding additional CPUs and memory. The other direction is to scale out, commonly described as scaling horizontally, which is to add additional servers to your environment.

Scale Up Scale Out

Components of a Rock Cluster

There are four major components of a Rock cluster. The following diagram outlines these components with further details below.

Components of a Rock Cluster



1. **Application Gateway**
   The application gateway's primary job is to direct traffic to the web sever cluster. It ensures that requests are load balanced over the various servers in the cluster. It also ensures that if a server goes down, that traffic is re-directed to other servers in the farm. The type and configuration of these gateways will vary depending on your cloud provider. For those using Azure, we recommend using the Azure App Gateway. One important configuration for the app gateway is that it should have session affinity enabled. This means that traffic for a specific client will always return to the same server. This is important, as Rock's check-in uses session state.

2. **Web Server Cluster**
   This is the group of web application servers that are participating in the cluster. More details on the configuration of these servers can be found below.

3. **Database**
   The database in a clustered environment isn't that different than one in a non-clustered environment. Some organizations choose to enable a database cluster, but this isn't required. The specifics of setting up your database will be unique to your cloud host. Whether your database is clustered or not is transparent to Rock.

4. **Message Bus**
   The message bus is what allows the various Rock web servers to communicate to other nodes in the cluster. The details of these conversations is discussed below. Two bus services are currently supported by Rock. They are:
   1. **Azure Message Bus**
   2. **Rabbit MQ**

> **Redis Server**
>
> Previous versions of Rock clusters relied on a Redis server. The message bus now replaces the need for the Redis server. We highly recommend moving away from the Redis solution to the new web farm features. Support for Redis clusters is deprecated as of Rock v13.

Benefits of Clusters

Server clusters have two primary benefits: increased scale and added redundancy. Let's consider each in turn.

Clusters obviously provide additional computing resources for supporting your digital strategy. Modern cloud environments have no problem providing large servers, but when hosting web applications, at a certain point the law of diminishing returns kicks in. In a Rock environment, CPU is often the constraint. Memory, though important, usually isn't nearly as limiting a factor as the CPU.

The second benefit of clustering is redundancy. While outages are rare and quickly recovered in a cloud environment, having multiple servers provides a failover capability. While this is a great benefit, be careful. The added complexity of a clustered environment can actually reduce the up-time of your application if it isn't well architected and managed by a team who understands the new infrastructure and services required.

Though not a primary benefit, some organizations use clusters to partition their traffic. This allows dedicated servers for traffic to support things like the internal portal, external websites, check-in and API hosting. By providing dedicated resources for these services, you can limit the spill-over of congestion from one service to another.

What Makes Clusters Difficult

Implementing a Rock cluster drastically increases the complexity of your environment. Below are several factors you should consider before moving forward with a cluster project.

1. **Increase Infrastructure Knowledge:** As you've seen, there are several new infrastructure components required to run a Rock cluster. Components like application gateways can be difficult to set up and support. Be sure that your team has the necessary engineering knowledge to support these environments now and in the future.
2. **Rock Updates:** Updating Rock is more difficult in a clustered environment. Currently, each server will need to be updated manually (or through custom dev ops automation).
3. **Rock File Types:** In a clustered environment, it's important that all Rock file types are cloud or database backed. If you use the 'File System' type, files will only be available on a single node in the cluster.
4. **File System Sync:** Besides file types there are other opportunities for files to be uploaded to a single node. For example, the HTML editor allows files to be uploaded to the ~/Content directory of a node. These files will appear as broken

links to clients pointed at other web application nodes in the cluster. There are tools you can configure to sync these directories, but they represent another service to configure and maintain.
5. **TLS Complexity:** In a clustered environment, we recommend you use a provisioned wildcard digital certificate instead of an ACME certificate. This represents an additional cost and you'll need to remember to renew it.
6. **Limited Partner Support:** There are a limited number of Rock Partners with familiarity with Rock's web farm features.

Recommendations for Rock Clusters

As you consider the design of your Rock cluster, there are several principles you should consider. Below are our recommendations to consider as you plan your implementation.

1. **The Best Cluster is No Cluster:** Simply put, server clusters are difficult and add complexity to the environment.
2. **Prefer Fewer Larger Servers:** As you plan your cluster, we recommend having fewer large nodes than more smaller nodes. The fewer nodes you have the less complexity you have.
3. **Scale Up Not Out:** If you would like to add additional resources for critical timeframes, we recommend you scale up the existing servers in your cluster rather than adding new nodes to the cluster. Adding new nodes requires complex dev ops to create node templates and deploy them as needed, whereas increasing the size of a running node is typically much easier using existing tools provided by your cloud service.

Configuring a Rock Web Farm

Before getting started you'll want to complete the following pre-steps:

1. **Web Farm License Key:** Running a web farm requires a unique license key from Spark. This is necessary as the web farm feature has an additional cost to help support the feature without the need to use funding from donations. Reach out to Spark at info@sparkdevnetwork.org to obtain information about how to license the web farm.
2. **Infrastructure Configuration:** The configuration steps below assume you have all of the needed infrastructure configured and Rock installed and running. It's outside of the scope of this document to provide step-by-step guidance on configuring the infrastructure, as these steps will be very different depending on your cloud hosting provider. Reach out to a Rock Partner (https://www.rockrms.com/partners) if you need help with this configuration. Below is a list of the high-level components that should be configured:
    1. App Gateway configured and working.
    2. Database server configured.
    3. Web application nodes configured and running.
    4. Rock installed on each web application node. Be sure that Rock jobs is only installed to run on a single node of your cluster. This is set in the server's web.config file.

With these steps complete, you're now ready to configure the web farm. Below are the steps you'll need to follow.

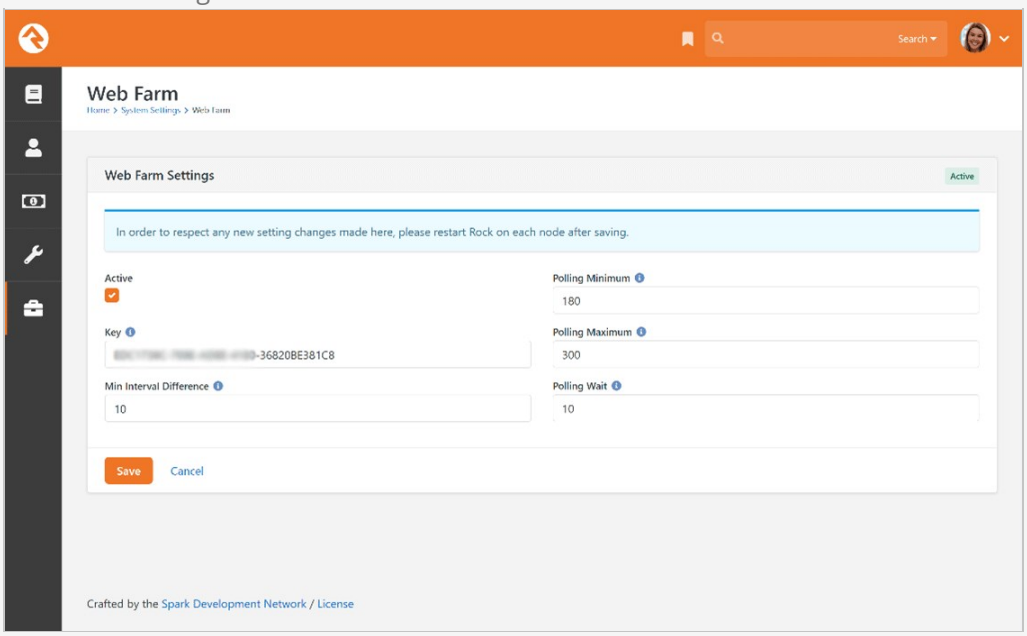
1. Configure the Message Bus

If you haven't yet configured a message bus in Rock, you'll need to do that before proceeding. See the Admin Hero Guide for more information on configuring the message bus.

2. Activate the Web Farm

Navigate to `Admin Tools > System Settings > Web Farm` to enable the web farm feature. This is where you will activate the web farm and enter your unique license key.

Web Farm Configuration

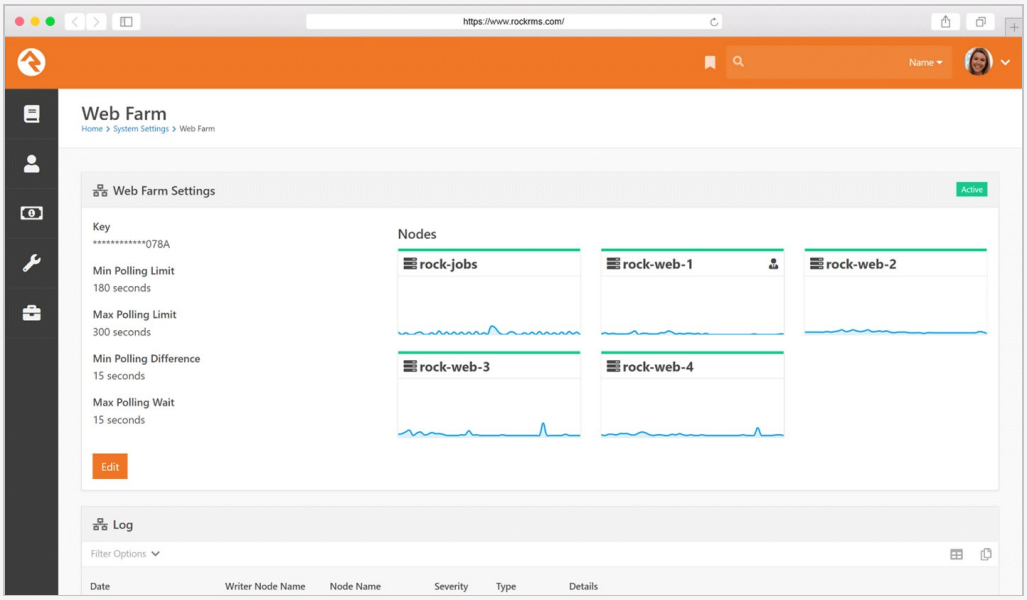


The remaining settings can be left as is. The additional configuration values are discussed below in the section on how nodes elect leaders.

3. Restart All Web Nodes

Once the configuration is set, you 'll need to restart all the nodes in the farm. Once restarted, the web farm is fully configured. You'll want to ensure that each node is displayed on the web farm page.

Web Farm Nodes



The Secret Life of Web Farm Nodes

Behind the scenes, the nodes of a web farm communicate with each other over the message bus to keep each other in the loop on what's happening inside the cluster. These messages are related to two primary types:

1. **Cache Invalidation:** When a node changes a value that lives inside of the Rock cache, it must notify the other nodes in the farm to also update their cache.
2. **Node Status:** As the nodes start-up, they elect a leader of the cluster. One of the roles of the leader is to check-in with each node on a routine basis to ensure it is still up and running. Any discrepancies the leader finds will be reported in the web farm log.

Node Start-up / Shutdown

As nodes start-up, they report their status to the web farm log and mark their status as active. They also report when they shutdown and change their status to inactive. This notifies the leader which nodes are expected to be running.

Leader Responsibilities

As noted above, one of the responsibilities of the node leader is to check that status of every node marked as being active. This status check is run on a configured polling schedule. On each run, the leader sends every active node a ping. If a node does not respond within a few seconds with a pong response, the leader reports the node as having been marked active but no longer responding.

Leader Election

You may be wondering how a leader is elected by the cluster. The election process is a sophisticated algorithm that allows the nodes to jointly elect a leader as well as trigger a re-election event, should the leader go AWOL. Let's take a closer look at how the election process works.

As a node starts-up, it randomly selects a polling interval to use. The selection of the random interval is guided by the configuration of the web farm. The configuration settings tell the node the minimum and maximum interval value. Using these values as boundaries, the node randomly selects a value somewhere between. It then checks the database to see if another node already has a similar interval (what constitutes a similar interval is also configurable). If a node is found with a similar interval it tries another random interval until an acceptable value is generated. This interval is then stored in the database for that specific node.

The leader of the cluster is the node with the smallest interval, meaning it will be the first to send out a request to other nodes for a status check. As each other node receives a status check request, it also notes that it has been contacted by the leader and therefore should reset its timer to ping other nodes. If the leader goes offline, the node with the next lowest interval will reach its status check interval and assume the leadership role. These changes of leadership are noted in the web farm log.

When a leader node comes back online, their smaller interval length will kick back in and it will regain leadership within a status cycle or two.

The current leader of a cluster is noted by a small icon in the upper right of the node card.

Leader Node

In the above scenario, the leader node is randomly selected. If you prefer, you can specify a particular node as leader by setting the interval value on the node's `WebFarmNode` record in the database. Setting the `ConfiguredLeadershipPollingIntervalSeconds` property disables the random value, which currently must be updated in SQL. Placing a value lower than the minimum will ensure that the node is always the leader when running. If that node goes offline, another node will still step in until it comes back online.

Node Names

By default, the name of the node will be read from the services machine name. If you prefer to use a different name, you can provide it in the web.config file with the app setting key of "NodeName".

Node Metrics

You'll notice that basic CPU metrics are provided for each node in the cluster. This provides a quick way to confirm that the load is being roughly balanced across your nodes. If you find that these charts vary greatly across your nodes, it may mean that something is not correctly configured in your application gateway.

Current Limitations

Rock's Web Farm features will continue to grow as more time and resources become available to fund resources. Below is a list of known limitations.

1. **Page Routes:** Changes and additions of page routes currently are not communicated to other nodes in the web farm. These are updated nightly when the AppPool restarts. If you need them sooner you'll need to restart each node in the cluster.
2. **Job Runner:** The current feature set will not prevent two servers from running jobs.